How to Reach Real-Time AI on Consumer Devices? Solutions for Programmable and Custom Architectures

(Invited Paper)

Stylianos I. Venieris†, Ioannis Panopoulos‡, Ilias Leontiadis†, Iakovos S. Venieris‡
†Samsung AI Center, Cambridge, UK, ‡National Technical University of Athens, Athens, Greece
Email: {s.venieris, i.leontiadis}@samsung.com, ioannispanop@mail.ntua.gr, venieris@cs.ece.ntua.gr

***Abstract*—The unprecedented performance of deep neural networks (DNNs) has led to large strides in various Artificial Intelligence (AI) inference tasks, such as object and speech recognition. Nevertheless, deploying such AI models across commodity devices faces significant challenges: large computational cost, multiple performance objectives, hardware heterogeneity and a common need for high accuracy, together pose critical problems to the deployment of DNNs across the various embedded and mobile devices in the wild. As such, we have yet to witness the mainstream usage of state-of-the-art deep learning algorithms across consumer devices. In this paper, we provide preliminary answers to this potentially game-changing question by presenting an array of design techniques for efficient AI systems. We start by examining the major roadblocks when targeting both programmable processors and custom accelerators. Then, we present diverse methods for achieving real-time performance following a cross-stack approach. These span model-, system- and hardware-level techniques, and their combination. Our findings provide illustrative examples of AI systems that do not overburden mobile hardware, while also indicating how they can improve inference accuracy. Moreover, we showcase how custom ASIC- and FPGA-based accelerators can be an enabling factor for next-generation AI applications, such as multi-DNN systems. Collectively, these results highlight the critical need for further exploration as to how the various cross-stack solutions can be best combined in order to bring the latest advances in deep learning close to users, in a robust and efficient manner.**

I. Introduction

The unprecedented predictive power of deep neural networks (DNNs) has led to their ever-increasing usage on mobile and embedded devices, transforming their capabilities and, consequently, our lives. At the same time, *real-time* AI applications are also gaining popularity. For instance, smart assistants are required to respond with low latency [1] while AI video upscaling algorithms are required to run at high frame rates in order to avoid rebuffering [2], [3].

Supporting real-time requirements on mobile hardware is becoming more and more challenging as the complexity of state-of-the-art DNNs is increasing exponentially [4]. Most device vendors have started incorporating System-on-Chips (SoCs) that can accelerate DNN computations, such as GPUs and NPUs. While these can significantly accelerate DNN inference, developers still face the same issue: *they have to support the wide variety of devices* that can be found in the wild [5]–[7]. This includes older devices, low- and mid-range smartphones, wearables and IoT devices. Hence, developers frequently resort to deploying simpler or heavily compressed CNNs at the expense of accuracy [8]. As real-time inference is not always possible, DNN developers also rely on fully or partially offloading to a remote infrastructure, such as the cloud or the edge [9], [10]. Offloading can improve inference latency and resolve the problem of wide device compatibility, at the expense of using network and cloud resources, raising privacy concerns [11] and yielding inconsistent user experience due to varying networking conditions [12].

While on-device optimisations and computation offloading can help supporting some real-time requirements, upcoming applications impose even stricter deadlines: self-driving cars need to process multi-sensor inputs within a few hundred milliseconds [13], an AR/VR headset typically performs scene recognition within 20 milliseconds while supporting 120 Hz frame-rates [14], whereas robotic assistants need to run multiple models simultaneously to achieve context awareness and to interact with their environment. Typically, these scenarios are only addressed by co-designing DNNs with domain-specific hardware, such as ASICs and FPGA-based accelerators.

In this paper, we will dive into prominent techniques that have been used to support real-time AI in both *general-purpose* and *customised* hardware platforms. We start by examining the major roadblocks and then present diverse methods for achieving real-time performance that span the whole stack: model-, system- and hardware-level techniques, and their combination. Moreover, we showcase how and under which settings custom ASIC- and FPGA-based accelerators can be an enabling factor for next-generation AI applications.

II. Common Roadblocks in Real-Time AI

In an AI system, a stream of input samples (*e.g.* photos, video frames, mic signals, or accelerometer readings) is processed by an AI model, typically a DNN, in order to perform an inference (*e.g.* object or speech recognition). Central to the operation of such a system is the hardware processing unit that executes the DNN inference. The architectural landscape of processing units for AI workloads can be classified into two main categories: *1)* programmable processors and *2)* custom accelerators. This classification is based on the *efficiency-flexibility trade-off* of the underlying hardware.

Despite the radical progress of deep learning, only a few big vendors have been in position to integrate state-of-the-art AI technologies across all their products. Even in these cases, a number of critical issues are challenging the efficient and wide integration of DNN-based algorithms in consumer devices:

1) ***DNN Diversity****:* DNN models vary in terms of task, architecture, workload and resource demands. These factors have a direct impact on the memory footprint, number of operations, computation-to-communication ratio, the parallelisation potential and the resilience to approximate computing techniques [15].

For classification tasks, even from 2012, DNNs such as AlexNet and VGG-16 exhibited orders of magnitude higher computational demands than other ML models. This was further aggravated with the development of large-scale models, such as ResNet-152 and DenseNet-161. Despite the design of efficient models, such as MobileNet and ShuffleNet, that employ novel blocks, such as depthwise separable convolutions, to reduce the number of operations, these blocks are often memory-bounded or underutilise the underlying processing hardware. As such, the theoretical complexity reduction does not always translate to actual performance gains upon deployment.

At the same time, tasks such as image/video super-resolution [16] and semantic segmentation [17], are characterised by even larger computational complexity. This mainly stems from the fact that, in contrast to classification DNNs that reduce the feature maps' size as we go deeper in the network, these tasks require the size of the feature maps to be maintained. The rationale behind this is that high-quality super-resolution or segmentation require the propagation of

information about high-frequency details, such as the texture or the contour of an object, until the output of the DNN. This property affects significantly both the memory footprint and the number of operations, imposing a barrier in achieving real-time performance.

In the field of NLP and ASR, applications are dominated by RNNs (*e.g.* LSTMs/GRUs) and Transformers. The primary computational challenge of these families of DNNs is that they consist of multiple matrix-vector multiplications and hence are memory-bounded. As a result, processors and accelerators that have typically been optimised for compute-bound convolutional layers and matrix-matrix multiplications are pushed to their limits [18] and performance becomes bounded by the available off-chip memory bandwidth [19]. The same holds for the case of Multi-Layer Perceptrons (MLPs) that rely only on the memory-bound fully-connected (FC) layers [18].

Recently, neural architecture search (NAS) methodologies [20] have rapidly been adopted to automatically generate highly accurate and, sometimes, compact models for a target task. Nonetheless, NAS often leads to nonintuitive topologies, up to the extreme case of randomly wired networks [21], [22]. The complex and irregular topology of such DNNs poses important problems in terms of both compiling them for existing programmable processors [23] and deriving a suitable custom accelerator [24].

In this context, the rapid algorithmic advancements from the AI community are in need for future-proof solutions and hence call for *general* hardware platforms that can be re-used from the following generations of DNNs. On the other hand, high performance often requires customisation, which in turn hurts generality. As a result, finding a balance between flexibility and customisation remains a challenging and crucial problem in the design of AI hardware.

*2) **Performance Objectives' Variability**:* Depending on the end application and target device, the performance requirements vary significantly in terms of accuracy, latency, throughput, energy and power across DNN applications. Even under the unified goal of real-time processing, the application determines the lowest acceptable accuracy and the platform dictates the available energy, power and resource budget of the system. For instance, interactive applications, such as VR and gaming, demand low latency (*e.g.* 20 ms), while wearable devices require ultra-low-power solutions (*e.g.* $<$1 W).

*3) **System Heterogeneity**:* The different processing capabilities of devices in the wild lead to wide system heterogeneity. This comprises both the system software and the underlying hardware. On the software side, the fragmented space of OS variants (*e.g.* numerous versions of Android, iOS, Tizen, *etc*), together with the partial support of a unified middleware (*e.g.* limited support and inconsistent performance of `NNAPI` across smartphones [6], [7], [25]), poses challenges in maintaining the functionality and performance through time and across devices. On the hardware side, the large number of vendors and the different use-cases have led to devices with broadly different characteristics [5]–[7], [25], such as processing capabilities, memory capacity, camera, mic and accelerometer sensors. As a result, performance cannot be trivially sustained across devices, leading to inconsistent quality of experience (QoE) for users of different devices.

*4) **Environment Dynamicity**:* Dynamicity is often manifested in the form of reduced processing speed, longer delays during memory transfers and degraded network bandwidth. The roots of this phenomenon stem from *i)* the multi-tasking nature of mobile systems [26], *ii)* the frequency throttling policies that are in-place to avoid overheating [27] and *iii)* the fluctuations in the quality of the network connectivity [28]. These factors often make the static design analysis and performance estimation futile, and necessitate the design of systems that can dynamically adapt to changes.

III. Real-Time AI on Programmable Processors

Consumer devices, such as smartphones and tablets, typically host processors that are able to serve a multitude of diverse workloads. As such, their design follows a more general-purpose approach and favours flexibility and programmability. We define as programmable processor any architecture that consists of processing elements that execute a stream of instructions, *without introducing domain-specific optimisations at the hardware or ISA level*.

Such processors span from ubiquitous mobile CPUs, such as Arm Cortex-A, Qualcom Kryo and Samsung Exynos [29], up to more specialised units, such as mobile GPUs, DSPs and NPUs. This class of processors can be found in many flavours, based on the performance needs of the application and the cost, power and form-factor constraints of the target platform. For instance, flagship smartphones tend to host more powerful CPUs (*e.g.* the Arm Cortex-X1 core in Samsung S21 Ultra) and GPUs than their mid- (*e.g.* Kryo 400 series in Samsung Galaxy A72) and low-tier (*e.g.* Arm Cortex-53 in Samsung Galaxy J7) counterparts. A similar situation can be observed for notebook and tablets which can host powerful processors with a medium power limit (*e.g.* Apple M1 on MacBook and iPad Pro with 15-watt TDP) compared to phones with tighter thermal limits (*e.g.* Apple A14 SoC with 5 TDP on iPhone 12). On the other hand, IoT devices, such as smart watches and home sensors, often rely on energy-efficient, but memory-constrained, microcontrollers (MCUs), so that they can be unintrusively integrated into the users' everyday life. Nonetheless, the extensive flexibility of programmable architectures comes at the cost of a hard limit on the attainable processing speed and energy efficiency [30].

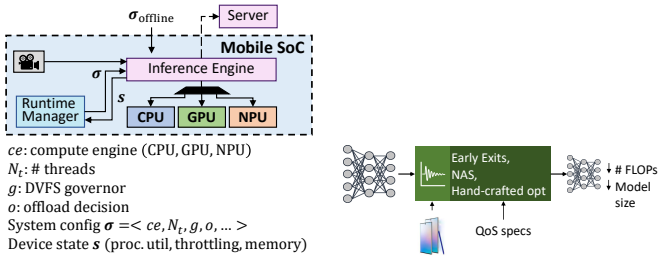
In the rest of this section, we present an array of solutions that make important strides towards real-time AI, highlighting the essential components to achieve this goal. We classify these solutions based on the entity of the system where the optimisation is implemented:

- System optimisations (Section III-A)
- Model optimisations (Section III-B)
- Joint model-system optimisations (Section III-C)

A. System Optimisations

An approach of addressing the system heterogeneity and meeting real-time performance for AI inference is to adapt the deployment to the characteristics of the device at hand. This adaptation process involves finding the highest-performing resource configuration of the target mobile SoC, such as enabling and disabling cores of different types, defining the task-to-processor mapping, setting the dynamic voltage and frequency scaling (DVFS) policy and making server offloading decisions (Fig. 1a). With the exception of dynamic DNNs [31], [32], the majority of deep learning models are characterised by a static workload which is known before run time. This advocates for an initial *static optimisation stage*. At the same time, modern consumer devices are increasingly dealing with concurrent execution of apps with various resource demands, performance needs and random arrival/completion times. As such, *dynamic adaptation* mechanisms are also key behind sustaining the required performance during DNN inference.

Static & Dynamic System Adaptation: OODIn [25] is an on-device framework that showcases the potential of system tuning to tailor the DNN inference to the target platform. To capture the multiple objectives of DNN inference workloads, OODIn introduces a multi-objective optimisation framework that combines resource constraints with accuracy and performance requirements. Next, the framework identifies key system parameters, including the task-to-processor mapping, the number of threads, the DVFS policy and



(a) (b)

Fig. 1: System tuning (a) and model adaptation (b), performed in isolation or jointly, constitute pillars of real-time AI.

the level of precision quantisation of the DNN model, and exposes them for optimisation to tailor the execution of the DNN to both the application-level performance needs and the underlying hardware. To this end, OODIn's workflow is divided into two components: the offline (or static) and the online (or dynamic).

During the offline stage, OODIn creates multiple model variants with different levels of quantisation in order to modify the accuracy-complexity trade-off of the user-supplied model. As such, OODIn's offline optimisation method takes into account both the model space and the user-supplied performance goals to yield the optimal model and system configuration. Static optimisation leads to average speedups of more than 70% over highly optimised status-quo implementations across diverse devices and DNN models.

On the other hand, the online phase is responsible for the mobile application's robustness and adaptability. OODIn tracks the mobile device's dynamic recourse availability changes, due to multi-tasking or thermal throttling, and reconfigures the selected parameters. Timely and efficient dynamic adaptation leads to latency reductions of up to 2.7× over statically optimised configurations.

Dynamic Onloading/Offloading: DNN developers who seek state-of-the-art performance and broad device compatibility, typically resort into offloading computation to a remote server, either on the cloud or the edge. While this can resolve the problem of supporting devices of various capabilities, cloud offloading can also result in high operation costs, privacy issues and excessive dependence on the networking conditions.

Computation onloading [33] aims to combine the best of both worlds: i) the cloud's elastic computational power and the ability to support a wide variety of devices and ii) the fact that modern embedded devices have, ever-increasing, DNN processing capabilities. The main idea is to split a DNN into two parts; during inference the device executes a part of the computation then transfers a heavily compressed version of the intermediate results to a powerful server to resume computation and then retrieve back the result. The main idea is to *onload as much computation as possible from cloud-native models into resource constrained devices* in order to maximise the overall performance and reduce the cloud cost, while meeting the application deadlines. As a result, powerful devices can process most of the DNNs locally and, therefore, save cloud resources, whereas less powerful devices might need more support from a server. These systems monitor and dynamically adjust the split point at run time, automatically freeing resources from the cloud by dynamically utilising on-device hardware. Results show that dynamic onloading can lead to an order of magnitude higher inference throughput while saving cloud resources.

B. Model Optimisations

Under settings where the underlying processing engine is assumed to be fixed, applying optimisations at the model level can lead to substantial gains (Fig. 1b). Well-investigated methods of reducing the cost of inference include quantisation [34], pruning [35] and low-rank factorisation [36]. Orthogonal to these methods, two prominent types of model optimisations that further push the performance on commodity processors constitute: *1)* hardware-aware model adaptation and *2)* hardware-agnostic efficient model design. Primary assumption in both cases is the availability of the training dataset for the target AI task, which enables the model-level modifications.

1) ***Hardware-aware Model Adaptation****:* Recently, a plethora of adaptive DNN architectures have been proposed. The overarching objective is to exploit the *variability in complexity* of different input samples in order to perform only the necessary amount of computation to obtain an accurate prediction. Moreover, this class of DNNs can tunably scale their resource usage and thus *dynamically adapt* to any fluctuations in resource availability, either due to thermal throttling or multi-tasking. To this end, various input-dependent execution mechanisms have been proposed, leading to dynamic, conditional DNN models. Such mechanisms include dynamically pruned DNNs [37] and early-exit models [38].

Hardware-aware Early-Exit DNNs: To extract peak performance, a stream of works has presented hardware-aware methods for the construction of early-exit DNNs [32]. Such frameworks consider the computational, memory and energy budget of a target platform, in order to strategically attach early exits across the depth of a given model and tune the associated early-exit policy.

HAPI [38] is a representative model-adaptation framework whose goal is to convert vanilla DNNs into high-performance early-exit models. This is achieved through a hardware-aware methodology that considers both the characteristics of the target platform and the maximum latency tolerance in order to automatically select the number and position of early exits along the DNN architecture. As such, the early-exit DNN topology is *statically* optimised before deployment. Then, *at run time*, HAPI adopts a tunable confidence-based early-exiting policy which dictates that a sample will stop at the first exit that yields a confident-enough prediction. Through this fine-grained parametrisation, HAPI tailors the early-exit model (number and placement of exits) and the early-exit policy (confidence threshold) to both the app-level performance requirements and the platform capabilities, resulting in 2.33× speedup and 2.53 percentage points (pp) higher accuracy than MobileNetV2 on Nvidia Jetson Xavier under the same 10-watt power budget, highlighting the gains that can be obtained through hardware-aware model adaptation.

2) ***Efficient Model Design****:* A promising approach that emphasises generality is the manual or automated design of efficient, lightweight models. Flows for efficient model design typically rely on platform-agnostic metrics, such as FLOP count and model size, to set a computational and memory budget. Although such proxy metrics often do not translate to actual processing gains [5], [39], notable performance gains have been achieved and mobile-friendly DNNs such as MobileNet [40], SqueezeNet [41] and EfficientNet [42] have been widely adopted in actual applications. Here, we describe three prominent approaches for designing efficient models.

Budget-aware Neural Architecture Search: Recently, significant effort has been placed into NAS (or AutoML) frameworks that aim to find high-accuracy models under computational or memory constraints [39], [43]. These frameworks typically adopt device-independent metrics to guide their search towards compact models that would potentially meet the required performance *across devices*.

Such a NAS-generated model is TPSR [44], a compact DNN for the task of image super-resolution. Optimised for perceptual quality and small footprint, TPSR delivers high-quality $\times 4$ image upscaling while consuming only 244 KB (FP32) or 61 KB (INT8) of memory. With an average latency of 71 ms per image (*i.e.* 14 frames-per-second) when upscaling to 720p using the NPU of a Qualcomm Snapdragon 865 SoC, TPSR showcases the potential of budgeted NAS even for the challenging case of mapping expensive tasks on smartphones and other resource-constrained IoT platforms.

AutoML-powered Model Compression: A drawback of running a complete NAS is the excessive computational requirement during the search phase. To alleviate this cost, it is possible to parametrise existing DNNs with parameters that expose an accuracy-complexity trade-off and exploit the efficacy of AutoML in order to find a high-performing configuration for these values. An example of this is ShrinkML [45], [46] which targets streaming LSTM-based models for automatic speech recognition (ASR) on mobile devices. ShrinkML employs low-rank factorisation of each layer in order to tunably prune the DNN weights. Each layer is compressed down to a different degree, with the per-layer compression ratio determined automatically using a reinforcement learning-based AutoML controller. This leads to a 17 ms latency on an Exynos 9810 CPU, corresponding to $3\times$ speedup over the vanilla model.

Hand-crafted Model Optimisation: A third approach for achieving real-time performance is to apply hand-engineered optimisations. Typically, such techniques are designed by domain experts and exploit domain-specific opportunities to improve the attainable performance. An instance of such a technique can be observed in the design of the bunched-LPCNet model [47] for Text-to-Speech (TTS) applications. The vanilla LPCNet is enhanced with *sample bunching*, a technique that allows it to produce more than one sample per inference and, in turn, reduce the overall computational cost. This is achieved by grouping together S temporally neighbouring samples and modifying the DNN architecture so that it can process all S samples as a bunch. Deployed on an Exynos 9820 CPU, bunched-LPCNet delivers a speedup of $2.19\times$ over the non-optimised model and achieves a real-time factor of 0.137. As such, by exploiting both the temporal nature of TTS and the large capacity of the LPCNet's GRUs, bunched-LPCNet demonstrates the gains that can be obtained through careful hand-crafted optimisations.

C. Joint Model-System Optimisation

A key approach to further boost the attainable performance is the joint optimisation of both the model architecture and the system parameters. Such schemes encompass techniques such as using alternative convolutional layers that map efficiently on the target hardware, designing multiple models and intelligently scheduling each input sample on the most suitable one based on a criterion, and strategically parallelising across the various processors of modern mobile SoCs.

Model Selection & Heterogeneous Computing: MobiSR [16], a framework for efficient super-resolution on smartphones, exemplifies the merits of model-system co-design. With super-resolution DNNs being especially computationally demanding, the proposed system introduces optimisations at various levels: exploiting the difference in upscaling difficulty among the different patches of an image, MobiSR uses a pair of models, each pinned to a different processor of the phone. The architecture of each model is optimised to yield efficient execution on the associated processor. At run time, each image patch's difficulty is quantified based on a total-variation metric and scheduled to the appropriate model-processor pair. Through this model-system co-optimisation, MobiSR delivers $4.79\times$ speedup over highly optimised single-processor implementations on a phone equipped with a Qualcomm Snapdragon 845 SoC.

Offloading Early-Exit DNNs for Robust Inference: Another approach that aims at both high performance and robust inference when the connectivity of the device to a server is uncertain is presented by SPINN [10]. SPINN combines distributed device-server inference with early-exit DNNs to deliver fast and robust inference across dynamic settings. The proposed system jointly and dynamically optimises the early-exit policy of the DNN (model-level optimisation) and the device-server partition point (system-level optimisation), providing previously unattainable adaptability to dynamic conditions. As such, SPINN achieves $2\times$ higher throughput over existing distributed inference systems that solely optimise system parameters. Moreover, by always placing an early exit on the device, the accuracy is maintained high even under severely constrained server availability. The concurrent use of distributed execution, adaptive early-exit DNNs and run-time system tuning leads to new levels of flexibility and enables deployment across diverse devices.

D. How personalised DNNs can help?

To be deployable in the wild, AI models need to generalise across a wide variety of inputs. For instance, facial landmark detectors are trained to capture various demographics, speech recognisers to accommodate different accents and voices, and home assistant robots to work reliably across diverse household configurations. Traditionally, to handle all these scenarios, parameter-heavy and computationally costly models are trained on massive datasets that aim to capture the majority of cases that will be encountered upon deployment. In contrast to this approach, a different paradigm introduces *on-device model personalisation*, aiming to tailor the DNN to a specific user or environment. Personalised models can be used not only to improve accuracy, but also as a way to improve efficiency.

One way to improve efficiency is to personalise early-exit DNNs [48]. On-device personalisation aims at producing classifiers along the depth of the network that are specialised for the user's data. At inference time, the model can either exit early if it is confident on its early output, or progressively refine the quality of the result using the deeper exits. A key advantage of early-exit personalisation is that training can take place even without ground-truth labels in a self-supervised manner, using the output of the DNN's last exit. This implies that a personalised task can become more and more efficient as more personalised inputs are available, without any user supervision. Furthermore, personalising only the early exits renders the training process lightweight enough to take place overnight, while the device is plugged in, without the need to access a remote server. This approach was demonstrated by PersEPhonEE [48]. By personalising an early-exit ResNet-50 using only on-device resources, PersEPhonEE achieved a $2.2\times$ speedup over the baseline model.

IV. Real-Time AI on Custom Accelerators

Towards extracting peak performance and attenuating the sources of inefficiency of standard processors, significant effort has been spent on designing accelerators for DNNs. We define as custom accelerators any architecture that applies domain-specific optimisations [49] and/or approximate computing techniques [15] to trade off lower programmability for higher performance. Such optimisations can target different components of the underlying hardware. Prominent instances constitute the following.

Simplified Control Logic: The programmable nature of processors requires the use of app-agnostic control logic, which is responsible for

tasks such as instruction fetching and accessing the register file. Instead, custom accelerators employ a range of techniques to minimise the overhead of this extraneous hardware or replace it with hardwired control. Broadly used techniques include *1)* domain-specific CISC ISAs and fusion of common operations [50]–[52] which amortise the overheads of instruction decoding over larger computational work, and *2)* data-driven streaming execution [53]–[55] where processing is triggered whenever data are fed to the accelerator. Such approaches have already been integrated in various accelerators, from Apple's M1 chip [56] and Nvidia's Tensor Cores [57] to mobile NPUs by Samsung [58], Qualcomm [59] and Huawei [60].

Specialized PE Design: Representative designs include, but are not limited to, PEs tailored for *i)* sparse DNNs employing zero-skipping units [61] (*e.g.* Samsung NPU [58]), *ii)* quantised DNNs through custom fixed- [62] (Qualcomm [59] and Samsung NPUs [58]) or floating-point representations (*e.g.* FP16 in Huawei Kirin NPUs [60] and `ms-fp9` in Microsoft's Brainwave NPU [19], two-precision [63], [64], mixed-precision [65] (*e.g.* Nvidia Tensor Cores [57], Qualcomm's 16-bit activations, 8-bit weights (A16W8) in Hexagon 698 NPU [59]) or bit-serial [66] units, and *ii)* binarised DNNs (BNNs) with dot-product units replaced with `popcount` operators [54].

Tailored Interconnection: The inter-PE and PEs-to-buffers interconnect is designed based on the workload of the target DNN [67], [68] for maximum performance and minimum external memory transfers. This is typically driven by the computation-to-communication ratio and the dimensions of the various layers of the target DNN.

Pipeline Organisation: This comprises accelerators [53], [54], [69] whose pipelines follow the topology either of the full DNN or of its main building block (*e.g.* residual block, Inception module, dense block, *etc*). This approach allows the fine-grained allocation of resources among the stages of the pipeline in order to match the processing rate of each stage and reach peak throughput. Similar designs can be found in various commodity devices, such as TV sets with custom AI upscaling processors [70].

Custom Memory Subsystem: The on-chip memory organisation is optimised to reduce the external memory bandwidth requirements and increase data-reuse. Such solutions typically restructure the on-chip memory and tailor the buffer sizes to match the DNN workload, while often introducing dedicated compression schemes for weights [36], [71]–[74] and feature maps [75], [76].

V. Looking Ahead: The Next Mile in AI Hardware

Custom hardware is in position to continue being a driving force in providing the computational power and energy efficiency needed for emerging AI-powered consumer platforms. In this section, we discuss two key directions for AI hardware architectures, namely *i)* multi-tenant AI accelerators for the concurrent execution of multiple DNNs and *ii)* automated model-hardware co-design methodologies for the joint optimisation of DNNs and hardware. Furthermore, we discuss how the unique properties of FPGAs can be the key in designing the next-generation of AI processors for consumer devices.

A. Multi-Tenant AI Systems

As the use of AI across applications and users increases, so do the computational demands. In this context, emerging systems tend to employ either pipelines of multiple DNNs or are required to serve queries from different users, each having their own dedicated DNN. This is especially important for inherently multi-tasking platforms, such as smartphones and home robots. However, existing platforms are optimised for the execution of single-DNN apps. Thus, to cope

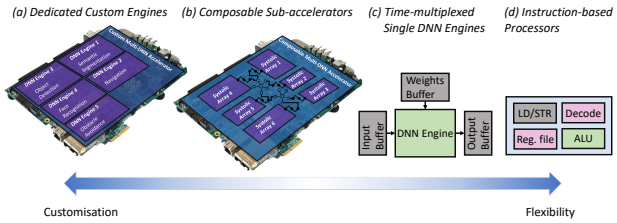


Fig. 2: Design space of multi-DNN accelerators.

with this increasing workload, new types of systems have to be developed, specifically optimised for multi-DNN settings.

Mapping multiple DNNs on a computing platform poses important challenges. With each DNN targeting a different task, the performance needs, such as throughput and latency, vary accordingly. This is aggravated by the fact that the multiple DNNs compete for the same pool of resources - off-chip bandwidth and on-chip computational and memory resources. As such, there is an emerging need for solutions that consider both the performance needs of each model and the resource constraints of the underlying platform. Recently, a few works have paved the way towards a new class of multi-DNN systems, encompassing both *1)* hardware and *2)* software aspects.

1) ***Multi-DNN Accelerators****:* Starting from 2018 [77], a number of accelerators [77]–[84] have focused on the multi-tenancy scenario. Fig. 2 shows the spectrum of multi-DNN hardware architectures. Key challenges comprise *i)* the customisation-programmability trade-off, *i.e.* how much to customise the hardware for each DNN and how much to reuse across DNNs, and *ii)* avoiding the resource contention between DNNs, *i.e.* how to best use the available resources without throttling the performance of the DNNs. The selected strategies for addressing these two issues determine to a great extent the design decisions of the underlying accelerator.

On the customisation side, f-CNN$^{\text{x}}$ [77] exploits the static workload of DNN models and derives *dedicated compute engines* for each DNN (Fig. 2a), highly tailored to the DNN's workload and application's performance needs. Furthermore, by means of a multi-DNN hardware scheduler, it optimises the external memory bandwidth sharing, in order to minimise the contention between the engines.

Focusing on flexibility, [78] introduces heterogeneous dataflow accelerators (HDAs), which consist of *multiple sub-accelerators* (Fig. 2b), each supporting a different dataflow. At run time, each DNN or each DNN layer can be mapped to the most suitable sub-accelerator. With the same goal of mapping each DNN layer to the most appropriate engine, `Planaria` [79] proposes the run-time construction of compute engines by means of multiple composable systolic arrays. Upon execution, the system examines the workloads of the target DNNs, appropriately connects the systolic arrays for each DNN layer and, finally, schedules execution.

With a focus on maximising the resource and bandwidth utilisation, AI-MT [80] co-locates multiple DNNs on *a single DNN engine* (Fig. 2c) and schedules simultaneously compute- and memory-bound sub-layers of the different DNNs. In this manner, the different sub-layers complementarily utilise the available computational and bandwidth resources, leading to high performance and efficient sharing of the proposed accelerator. Similarly, [81] and [85] also target single DNN engines and present dataflow mirroring and a preemption module, respectively, two hardware-level enhancements that aim to optimise the concurrent execution of multiple co-located DNNs on the underlying engine.

Another stream of work investigated the optimal derivation of multi-DNN architectures and the scheduling of DNNs on them

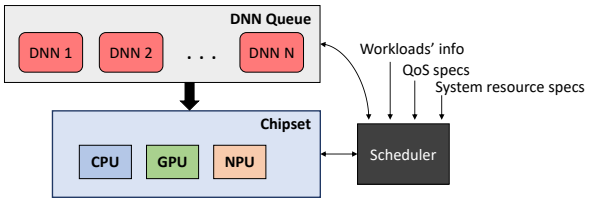


Fig. 3: System software for multi-DNN systems.

through design space exploration [77], [82] and contention-aware performance estimation techniques [83].

*2) **Multi-DNN System Software**:* To support multi-DNN application on existing and emerging hardware platforms, a number of software runtimes have been proposed. So far, research effort has been invested on optimising multi-DNN applications for programmable processors (Fig. 2d). In these works, the *scheduler* (Fig. 3) constitutes the most prominent component that effectively determines the attainable performance of the system. DART [86] is a scheduler that employs pipelining and priority-based scheduling across heterogeneous processors in order to execute multiple DNN tasks with deterministic response times. PREMA [85] proposes a DNN-specific priority-based preemptive scheduling algorithm to optimise the execution of multiple models on a single NPU. Similar to AI-MT [80], but from a scheduling perspective, *Layerweaver* [87] introduces a scheme for scheduling together a memory-bound and a compute-bound DNN, improving the utilisation of both the external memory bandwidth and the computational resources. Adopting a different viewpoint, MASA [88] comprises a memory-aware scheduler for minimising the memory swapping between DNNs. With a more model-software co-design approach, NestDNN [26] employs multi-capacity models that can dynamically adapt their computational needs. As such, a scheduler can adapt at run time the complexity of each DNN to optimise the overall multi-DNN execution. Finally, targeting mobile robot and IoT platforms, Lee *et al.* [89] proposed a weights virtualisation scheme that enables the sharing of weights among DNNs and their efficient in-memory execution.

*3) **Open Challenges**:* Here, we discuss open issues and future directions that have only lightly been explored by the initial efforts.

Performance vs. Flexibility: The early work on multi-DNN systems has currently produced diverse designs with a mostly decoupled consideration of the hardware and software aspects. Currently, peak performance is reached through fine-grained customisability [77] at the expense of a new hardware design cycle whenever a different set of DNNs is targeted. Although this approach may be viable for reconfigurable FPGA-based platforms, where the fabric can be reprogrammed with a different design in the occurrence of a new set of DNNs, ASIC designs require future-proof solutions that can amortise the fabrication cost through broad and efficient re-use across DNN workloads. This performance-flexibility gap is yet to be bridged in the multi-DNN context and remains a promising research avenue.

Approximate Computing for Multiple DNNs: Another promising approach for exposing more optimisation opportunities for multi-DNN accelerators is approximate computing. Under such schemes, the system would exploit performance-resource usage trade-offs with a controlled drop in accuracy [15]. Examples of such techniques include using different arithmetic precision for each DNN [64] or compressing their weights to a nonuniform degree [74]. For multiple DNNs, this encompasses the development of methods that exploit the cross-DNN redundancy, identify workload commonalities or differences in resilience to quantisation across the DNN models in order to reduce the external memory bandwidth requirements, better coordinate execution and allocate resources among the DNNs. An early approach was presented in [84] targeting multi-LSTM applications. In this case, the approximate computing method consists of a parametrised scheme for jointly decomposing the weight matrices of all the target LSTM models, followed by structured pruning and quantisation steps. The design of the associated accelerator is co-optimised together with approximation parameters in order to yield a tailored hardware design that satisfies a user-defined accuracy constraint, leading to 3×-5× speedup.

Multi-DNN Model-Hardware Co-Design: Finally, towards extracting both maximum performance and accuracy, model-hardware co-design approaches can be developed that would provide maximal degrees of freedom in the design space. Such methodologies can consider the multiple AI tasks and design from scratch both the DNN architectures and the underlying hardware. An early work towards this direction is ASICNAS [90]. To tackle the exponential design space of multi-DNN and accelerator co-optimisation, ASICNAS considers a limited number of pre-defined hardware architectures in its search space. With more than 2× energy savings and less than 1.6% accuracy drop, this work showcases the potential of co-design schemes in pushing further the performance of multi-DNN systems. Nevertheless, the primary challenge that obstructs multi-DNN model-hardware co-design is still present: the excessively high-dimensional design space that includes model-, scheduling- and hardware-level parameters. As such, research effort needs to be invested in overcoming this complexity through efficient methodologies in order to lead to the next-generation of multi-DNN platforms.

B. Automated Model-Hardware Co-Design

Traditional flows in the development of AI products consist of two steps: *1)* designing and training a DNN model that achieves the required accuracy for the target task under a FLOPs or memory budget and *2)* optimising the resulting model for execution on a target platforms, *e.g.* particular mobile phones and IoT devices. In spite of each successes, this approach can lead to suboptimal performance.

An alternative single-stage paradigm that is gaining traction is to *jointly* search for the DNN architecture and the hardware design [91]–[97]. Such a co-design approach can lead to closer-to-optimal configurations and aims to deliver peak performance in terms of both accuracy and processing speed. Nevertheless, main barrier constitutes the excessively large model-hardware design space.

To counteract the complex design space and explore a sufficiently large number of candidate designs, one line of work [91], [93]–[95], [98] has adopted pre-defined hardware templates and expose only high-level design parameters in the search space. Others works have incorporated streaming architectures with finer-grained customisability in their hardware design space [96] or have integrated quantisation into the search space [94], [96]

In an endeavour to push the hardware efficiency to its limits, recent works [99], [100] have designed DNN models that map well to FPGA building blocks. For instance, LUTNet [99] and LogicNets [100] incorporate Look-Up Tables (LUTs) as their primitive computational unit, reaching substantial area reduction and throughput gains over both conventional and binarised NNs. The resulting models can be directly mapped to FPGA-based platforms, avoiding the source of inefficiencies of more generic architectures. This is especially important for very resource-constrained platforms in IoT use-cases, where low-cost FPGAs without explicit DSP blocks are often deployed. Nonetheless, with this technology being at its infancy, the high performance currently comes with non-negligible drop in accuracy, which in some applications cannot be tolerated. As such, to incentivise

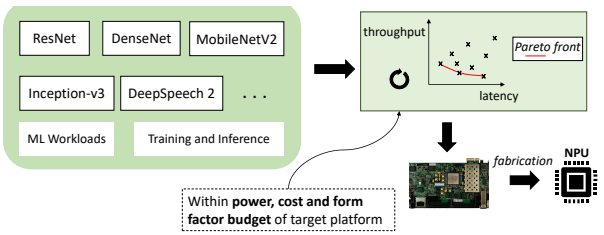


Fig. 4: FPGA-enabled exploration of next-generation AI processor architectures.

the wider exploration and potentially real-world adoption of these approaches, the performance and accuracy of such designs has to be scaled up and demonstrated on broader use-cases.

C. FPGAs for Deriving Next-Generation AI Processors

At the moment, there is a constant trend towards integrating NPUs into both mobile SoCs [5], [6] and servers [18], [19]. At the same time, deep learning models are evolving rapidly, with architectural changes affecting also their computational characteristics. Due to this, coming up with energy-efficient and high-performance accelerator designs becomes a challenge. In this context, FPGAs can be a key enabler in discovering future NPU designs (Fig. 4). By exploiting the reconfigurability of FPGAs, a large number of candidate hardware designs can be explored and run on the FPGA platform to measure critical metrics, including processing speed, power consumption and area. Given the constraints of the target platform across these dimensions, the objective of this process is to find the Pareto-optimal accelerator design for a number of representative DNN models. After the highest performing design has been identified, it can be converted to an ASIC and integrated as an NPU into future consumer devices.

VI. Conclusion

As real-time AI applications are becoming more and more popular, their use-cases are also becoming more demanding. Supporting such applications on mobile and embedded hardware that is ubiquitous across consumer devices poses important challenges. In this paper, we looked into the current roadblocks that need to be addressed and identified key themes such as the DNN and hardware heterogeneity as well as the dynamicity of the execution environment. Afterwards, we looked into state-of-the-art practices and research directions for both programmable processors and custom accelerators. We further highlighted important future research avenues, with emphasis on multi-tenant inference systems and model-hardware co-design. Our findings reinforce the need to provide solutions across the whole stack; combined research on model, system, platform and hardware optimisations will be of key importance in order to support the next generation of real-time AI applications on mobile/embedded devices.

References

[1] R. Vipperla *et al.*, "Learning to Listen... On-Device: Present and Future Perspectives of On-Device ASR," *GetMobile*, 2020.

[2] Nvidia, "Dynamic Super-Resolution Improves Your Games with 4K-Quality Graphics on HD Monitors," https://www.nvidia.com/en-us/geforce/news/dynamic-super-resolution-instantly-improves-your-games-with-4k-quality-graphics/, 2021, accessed: June 30, 2021.

[3] R. Lee, S. I. Venieris, and N. D. Lane, "Deep Neural Network-based Enhancement for Image and Video Streaming Systems: A Survey and Future Directions," *ACM Comput. Surv.*, 2021.

[4] X. Xu, Y. Ding, S. Xiaobo Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, "Scaling for edge inference of deep neural networks," *Nature Electronics*, vol. 1, 04 2018.

[5] M. Almeida *et al.*, "EmBench: Quantifying performance variations of deep neural networks across modern commodity devices," in *EMDL*, 2019.

[6] A. Ignatov *et al.*, "AI Benchmark: All About Deep Learning on Smartphones in 2019," in *ICCVW*, 2019.

[7] C. Wu *et al.*, "Machine Learning at Facebook: Understanding Inference at the Edge," in *HPCA*, 2019.

[8] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the State of Neural Network Pruning?" in *MLSys*, 2020.

[9] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *ASPLOS*.

[10] S. Laskaridis *et al.*, "SPINN: Synergistic Progressive Inference of Neural Networks over Device and Cloud," in *MobiCom*, 2020.

[11] F. Mo *et al.*, "DarkneTZ: Towards Model Privacy at the Edge Using Trusted Execution Environments," in *MobiSys*, 2020.

[12] Yuan Zhang, Hao Liu, Lei Jiao, and Xiaoming Fu, "To offload or not to offload: An efficient code partition algorithm for mobile cloud computing," in *CLOUDNET*, 2012.

[13] S.-C. Lin *et al.*, "The Architectural Implications of Autonomous Driving: Constraints and Acceleration," in *ASPLOS*, 2018.

[14] L. Liu *et al.*, "Cutting the Cord: Designing a High-Quality Untethered VR System with Low Latency Remote Rendering," in *MobiSys*, 2018.

[15] E. Wang, J. J. Davis, R. Zhao, H.-C. Ng, X. Niu, W. Luk, P. Y. K. Cheung, and G. A. Constantinides, "Deep Neural Network Approximation for Custom Hardware: Where We've Been, Where We're Going," *ACM Comput. Surv.*, 2019.

[16] R. Lee, S. I. Venieris, L. Dudziak, S. Bhattacharya, and N. D. Lane, "MobiSR: Efficient On-Device Super-Resolution through Heterogeneous Mobile Processors," in *MobiCom*, 2019.

[17] A. Kouris, S. I. Venieris, S. Laskaridis, and N. D. Lane, "Multi-Exit Semantic Segmentation Networks," in *arXiv*, 2021.

[18] N. P. Jouppi *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *ISCA*, 2017.

[19] J. Fowers *et al.*, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," in *ISCA*, 2018.

[20] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," in *ICLR*, 2017.

[21] S. Xie, A. Kirillov, R. Girshick, and K. He, "Exploring Randomly Wired Neural Networks for Image Recognition," in *ICCV*, 2019.

[22] M. Wortsman, A. Farhadi, and M. Rastegari, "Discovering Neural Wirings," in *NeurIPS*, 2019.

[23] B. H. Ahn *et al.*, "Ordering Chaos: Memory-Aware Scheduling of Irregularly Wired Neural Networks for Edge Devices," in *MLSys*, 2020.

[24] R. Kuramochi and H. Nakahara, "An FPGA-Based Low-Latency Accelerator for Randomly Wired Neural Networks," in *FPL*, 2020.

[25] S. I. Venieris, I. Panopoulos, and I. S. Venieris, "OODIn: An Optimised On-Device Inference Framework for Heterogeneous Mobile Devices," in *IEEE SMARTCOMP*, 2021.

[26] B. Fang *et al.*, "NestDNN: Resource-Aware Multi-Tenant On-Device Deep Learning for Continuous Mobile Vision," in *MobiCom*, 2018.

[27] A. K. Singh *et al.*, "Dynamic Energy and Thermal Management of Multi-core Mobile Platforms: A Survey," *IEEE Design Test*, 2020.

[28] A. Cartas *et al.*, "A Reality Check on Inference at Mobile Networks Edge," in *EdgeSys*, 2019.

[29] B. Grayson *et al.*, "Evolution of the Samsung Exynos CPU Microarchitecture," in *ISCA*, 2020.

[30] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding Sources of Inefficiency in General-Purpose Chips," in *ISCA*, 2010.

[31] H. Shen *et al.*, "Nimble: Efficiently Compiling Dynamic Neural Networks for Model Inference," in *MLSys*, 2021.

[32] S. Laskaridis, A. Kouris, and N. D. Lane, "Adaptive Inference through Early-Exit Networks: Design, Challenges and Directions," in *EMDL*, 2021.

[33] M. Almeida, S. Laskaridis, S. I. Venieris, I. Leontiadis, and N. D. Lane, "DynO: Dynamic Onloading of Deep Neural Networks from Cloud to Device," in *arXiv*, 2021.

[34] B. Jacob *et al.*, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *CVPR*, 2018.

[35] T.-J. Yang *et al.*, "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications," in *ECCV*, 2018.

[36] A. Kouris, S. I. Venieris, M. Rizakis, and C.-S. Bouganis, "Approximate LSTMs for Time-Constrained Inference: Enabling Fast Reaction in Self-Driving Cars," *IEEE Consumer Electronics Magazine*, 2020.

[37] X. Gao, Y. Zhao, Ł. Dudziak, R. Mullins, and C.-z. Xu, "Dynamic Channel Pruning: Feature Boosting and Suppression," in *ICLR*, 2018.

[38] S. Laskaridis, S. I. Venieris, H. Kim, and N. D. Lane, "HAPI: Hardware-Aware Progressive Inference," in *ICCAD*, 2020.

[39] L. Dudziak, T. Chau, M. Abdelfattah, R. Lee, H. Kim, and N. Lane, "BRP-NAS: Prediction-based NAS using GCNs," in *NeurIPS*, 2020.

[40] M. Sandler *et al.*, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *CVPR*, 2018.

[41] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5 MB model size," *arXiv*, 2016.

[42] M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *ICML*, 2019.

[43] A. Gordon *et al.*, "MorphNet: Fast Simple Resource-Constrained Structure Learning of Deep Networks," in *CVPR*, 2018.

[44] R. Lee, Ł. Dudziak, M. Abdelfattah, S. I. Venieris, H. Kim, H. Wen, and N. D. Lane, "Journey Towards Tiny Perceptual Super-Resolution," in *ECCV*, 2020.

[45] Łukasz Dudziak, M. S. Abdelfattah, R. Vipperla, S. Laskaridis, and N. D. Lane, "ShrinkML: End-to-End ASR Model Compression Using Reinforcement Learning," in *Interspeech*, 2019.

[46] A. Mehrotra *et al.*, "Iterative Compression of End-to-End ASR Model Using AutoML," in *Interspeech*, 2020.

[47] R. Vipperla *et al.*, "Bunched LPCNet: Vocoder for Low-Cost Neural Text-To-Speech Systems," in *Interspeech*, 2020.

[48] I. Leontiadis, S. Laskaridis, S. I. Venieris, and N. D. Lane, "It's Always Personal: Using Early Exits for Efficient On-Device CNN Personalisation," in *HotMobile*, 2021.

[49] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions," *ACM Comput. Surv.*, 2018.

[50] S. Liu *et al.*, "Cambricon: An Instruction Set Architecture for Neural Networks," in *ISCA*, 2016.

[51] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-Layer CNN Accelerators," in *MICRO*, 2016.

[52] Y. Xing *et al.*, "DNNVM: End-to-end compiler leveraging heterogeneous optimizations on FPGA-based CNN accelerators," *TCAD*, 2020.

[53] S. I. Venieris and C.-S. Bouganis, "fpgaConvNet: Mapping Regular and Irregular Convolutional Neural Networks on FPGAs," *TNNLS*, 2019.

[54] Y. Umuroglu *et al.*, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *FPGA*, 2017.

[55] S. I. Venieris and C.-S. Bouganis, "Latency-Driven Design for FPGA-based Convolutional Neural Networks," in *FPL*, 2017.

[56] Apple, "Apple M1," https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/, 2020, accessed: June 30, 2021.

[57] J. Appleyard and S. Yokim, "Programming Tensor Cores in CUDA 9," October 2017, [Online; posted 17-October-2017]. [Online]. Available: https://devblogs.nvidia.com/programming-tensor-cores-cuda-9/

[58] J.-W. Jang *et al.*, "Sparsity-Aware and Re-configurable NPU Architecture for Samsung Flagship Mobile SoC," in *ISCA*, 2021.

[59] Qualcomm, "Snapdragon Neural Processing Engine," https://developer.qualcomm.com/docs/snpe/snapdragon_npe_runtime.html, 2021, accessed: June 30, 2021.

[60] H. Liao, J. Tu, J. Xia, and X. Zhou, "DaVinci: A Scalable Architecture for Neural Network Computing," in *HotChips*, 2019, pp. 1–44.

[61] J. Albericio *et al.*, "Bit-Pragmatic Deep Neural Network Computing," in *MICRO*, 2017.

[62] A. Rajagopal, D. Vink, S. Venieris, and C.-S. Bouganis, "Multi-Precision Policy Enforced Training (MuPPET) : A Precision-Switching Strategy for Quantised Fixed-Point Training of CNNs," in *ICML*, 2020.

[63] A. Kouris, S. I. Venieris, and C.-S. Bouganis, "CascadeCNN: Pushing the Performance Limits of Quantisation in Convolutional Neural Networks," in *FPL*, 2018.

[64] ——, "A Throughput-Latency Co-Optimised Cascade of Convolutional Neural Network Classifiers," in *DATE*, 2020.

[65] H. Sharma *et al.*, "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network," in *ISCA*, 2018.

[66] P. Judd *et al.*, "Stripes: Bit-Serial Deep Neural Network Computing," in *MICRO*, 2016.

[67] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated Systolic Array Architecture Synthesis for High Throughput CNN Inference on FPGAs," in *DAC*, 2017.

[68] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects," in *ASPLOS*, 2018.

[69] Huimin Li *et al.*, "A High Performance FPGA-based Accelerator for Large-Scale Convolutional Neural Networks," in *FPL*, 2016.

[70] Samsung, "AI Upscaling on Samsung TVs," https://www.samsung.com/au/support/tv-audio-video/ai-upscaling-on-samsung-tvs/, 2020, accessed: June 30, 2021.

[71] S. Han *et al.*, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *ISCA*, 2016.

[72] C. Ding *et al.*, "CirCNN: Accelerating and Compressing Deep Neural Networks Using Block-Circulant Weight Matrices," in *MICRO*, 2017.

[73] C. Deng *et al.*, "PermDNN: Efficient Compressed DNN Architecture with Permuted Diagonal Matrices," in *MICRO*, 2018.

[74] S. I. Venieris, J. Fernandez-Marques, and N. D. Lane, "unzipFPGA: Enhancing FPGA-based CNN Engines with On-the-Fly Weights Generation," in *FCCM*, 2021.

[75] Y. Chen, T. Yang, J. Emer, and V. Sze, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," *JETCAS*, 2019.

[76] A. Montgomerie-Corcoran and C.-S. Bouganis, "DEF: Differential Encoding of Featuremaps for Low Power Convolutional Neural Network Accelerators," in *ASP-DAC*, 2021.

[77] S. I. Venieris and C.-S. Bouganis, "f-CNNx: A Toolflow for Mapping Multiple Convolutional Neural Networks on FPGAs," in *FPL*, 2018.

[78] H. Kwon *et al.*, "Heterogeneous Dataflow Accelerators for Multi-DNN Workloads," in *HPCA*, 2021.

[79] S. Ghodrati *et al.*, "Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks," in *MICRO*, 2020.

[80] E. Baek, D. Kwon, and J. Kim, "A Multi-Neural Network Acceleration Architecture," in *ISCA*, 2020.

[81] J. Lee, J. Choi, J. Kim, J. Lee, and Y. Kim, "Dataflow Mirroring: Architectural Support for Highly Efficient Fine-Grained Spatial Multitasking on Systolic-Array NPUs," in *DAC*, 2021.

[82] R. Kedia, S. Goel, M. Balakrishnan, K. Paul, and R. Sen, "Design Space Exploration of FPGA Based System with Multiple DNN Accelerators," *IEEE Embedded Systems Letters*, 2020.

[83] S. Goel, R. Kedia, M. Balakrishnan, and R. Sen, "INFER: INterFerence-aware Estimation of Runtime for Concurrent CNN Execution on DPUs," in *ICFPT*, 2020.

[84] S. Ribes, P. Trancoso, I. Sourdis, and C.-S. Bouganis, "Mapping Multiple LSTM models on FPGAs," in *ICFPT*, 2020.

[85] Y. Choi and M. Rhu, "PREMA: A Predictive Multi-Task Scheduling Algorithm for Preemptible Neural Processing Units," in *HPCA*, 2020.

[86] Y. Xiang and H. Kim, "Pipelined Data-Parallel CPU/GPU Scheduling for Multi-DNN Real-Time Inference," in *RTSS*, 2019.

[87] Y. H. Oh *et al.*, "Layerweaver: Maximizing Resource Utilization of Neural Processing Units via Layer-Wise Scheduling," in *HPCA*, 2021.

[88] B. Cox, J. Galjaard, A. Ghiassi, R. Birke, and L. Y. Chen, "Masa: Responsive Multi-DNN Inference on the Edge," in *PerCom*, 2021.

[89] S. Lee and S. Nirjon, "Fast and Scalable In-Memory Deep Multitask Learning via Neural Weight Virtualization," in *MobiSys*, 2020.

[90] L. Yang *et al.*, "Co-Exploration of Neural Architectures and Heterogeneous ASIC Accelerator Designs Targeting Multiple Tasks," in *DAC*, 2020.

[91] C. Hao *et al.*, "FPGA/DNN Co-Design: An Efficient Design Methodology for IoT Intelligence on the Edge," in *DAC*, 2019.

[92] C. Hao, Y. Chen *et al.*, "NAIS: Neural Architecture and Implementation Search and its Applications in Autonomous Driving," in *ICCAD*, 2019.

[93] M. S. Abdelfattah, Ł. Dudziak, T. Chau, R. Lee, H. Kim, and N. D. Lane, "Best of Both Worlds: AutoML Codesign of a CNN and its Hardware Accelerator," in *DAC*, 2020.

[94] W. Jiang *et al.*, "Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start," *TCAD*.

[95] L. Yang *et al.*, "Co-exploring neural architecture and network-on-chip design for real-time artificial intelligence," in *ASP-DAC*.

[96] Z. Dong *et al.*, "HAO: Hardware-aware Neural Architecture Optimization for Efficient Inference," in *FCCM*, 2021.

[97] K. Choi, D. Hong, H. Yoon, J. Yu, Y. Kim, and J. Lee, "DANCE: Differentiable Accelerator/Network Co-Exploration," in *DAC*, 2021.

[98] W. Jiang *et al.*, "Hardware/Software Co-Exploration of Neural Architectures," *TCAD*, 2020.

[99] E. Wang, J. J. Davis, P. Y. K. Cheung, and G. A. Constantinides, "LUTNet: Learning FPGA Configurations for Highly Efficient Neural Network Inference," *TC*, 2020.

[100] Y. Umuroglu *et al.*, "LogicNets: Co-Designed Neural Networks and Circuits for Extreme-Throughput Applications," in *FPL*, 2020.